Intelligent container for Deep learning model

Raj Marri
*rmwwc@mail.umkc.edu

***Abstract*—Deployment of deep learning model into production from a development environment is quite a complex affair; choosing the cost and performance-optimized cloud provided services; implementation of streaming or batch training strategies; security hardening of the model; portability of the model across environments and cloud platform are some of the major challenges. Containerized deployment of deep learning models has gained immense traction across industries. To meet demand, traditional cloud providers offer ML build and deployment services like AWS Sagemaker and Azure Machine learning studio, which limit the freedom of choosing a user's favorite framework and performance after deployed. There does not exist any mechanism with which one can compare and contrast the capabilities of containers across different frameworks and providers and, provision transfer learning and domain adaptation capabilities for the model . We propose a novel intelligent container which can dynamically inject dependencies for hosting deep learning model across public cloud platforms for easy and high-performance deployment of the deep learning model. We make use of AI to automate the dependency injection into a container to make it dynamic to achieve optimized, high performance, simple deployment of the deep learning model.**

***Index Terms*—public cloud, deep leaning, stream and batch models, cost optimisation.**

I. Introduction

Most industries have started adopting hybrid clouds to benefit from easy of deployment in to production. ITaaS [1] or ML as a service (Machine Learning as a Service) plays major role in cost and performance optimisation for the of workloads with underpinning infrastructure. Public cloud adoption has increased drastically in the recent years. Container-based virtualization is becoming more popular than virtual ma- chines for deploying applications, especially those that can be designed and implemented as stateless and immutable micro-services [3]. This is because the former is simpler and considerably more lightweight than the latter [4], even as the latter is mature, and more secure.If one aims to offer a container based SaaS solution hosted on a hybrid cloud, the hosting complexity can be high.

Domain adaptation algorithms start with the homogeneous settings, i.e., data in both the source domain and the target domain are sampled from the same type of feature. However, one cannot always find a source domain which happens to be sampled with exactly the same type of feature with the target domain, and it is hasty to arbitrarily sample the target domain with the same feature of the available source domain because each feature has its pros and cons. Typically, domain adaptation deals with the problem, where a well-labeled source domain and an unlabeled target domain are involved, and the two domains have divergent probability distributions. It aims to leverage the label information in the source domain, so that the task in the target domain can be solved by knowledge transfer [8]. As a practical branch of transfer learning [8], domain adaptation has been exploited in many fields, e.g., image classification , objection recognition [9], [10], text categorization [7], [11], and video event detection [12]. Stateless microservice design translates to better reliability since immutable software does not need to shoulder the burden of handling session affinity [7]. Due to this, containerized applications that are designed to be stateless are better placed to exploit the elasticity and scalability of the cloud. For example, if an application is designed as a set of stateless microservices, a load balancer can instantly spawn a new containerized microservice instance to replace a failed one. However, while the microservice architecture generally translates to higher availability, performance can potentially be at a premium. We are going to adopt microservices architecuture for our approach.

Our intelligent container is aware of the cloud services through a common catalogue that computes the best price for hosting, provides a efficient lookup of available models for transfer learning and domain adoption.The advancements in container engines lead to the appearance of multiple management frameworks. Peinl et. al. [5] stated that a container management framework is responsible for many tasks such as scheduling, performance monitoring, load balancing, migration, service discovery, networking, etc. Also, they conducted a detailed comparison of several container management frameworks and specified a taxonomy for their features. They carried out a comparison between various tools, stacks, and integration techniques, and showed solutions for their integration issues. They concluded that not all container features are currently fulfilled but the appearance of new tools and frameworks is deriving the market to maturity.

II. Related Work

[13] focus on the conventional situation, in which only two domains, one in source and one in target, are involved, and data from both domains are sampled from the same features. However, one cannot always find a source domain which happens to be represented by the same features with the target domain.[13] mainly focused on the containeration services for the application across public cloud providers and, there research is only focused on applications not deep learning models.The work by [18] on scalability analysis of containers reported in the recent past reinforces the need for TQ and RQ in the semantics of container fitness. describes container selection policies that can increase conformance to service level commitment. This approach can be extended to the deployment of the deep learning model containers for

cost effective utilisation of public cloud. examines anomaly detection of processes in Linux containers. This approach can be used to add the cyber security aspect to the deep learning model to detect any process anomalies that are observed.[11] work has reported that the learning algorithm is only one component of a machine learning platform that represents a small fraction of the code[18] Data and model parallelism require distributed systems and orchestration that exceed capabilities of many single-machine solutions. [20] proposed new acceleration engines, such as DeepEar and DeepX, to support different deep learning applications in the latest mobile systems on chips. [19] Introducing deep learning into more IoT applications is another important research issue. Our approach can be extended to the IOT edge devices. [22] develop three methods two kinds of neural nets and texture code cue. Also, their combination was used, and the results were compared, the best performing method was the neural net cue.[25] transfer learning is proposed to handle challenges by borrowing knowledge from other external well-labeled data . [23] dictionary learning scheme manages to construct a single shared dictionary or multiple common dictionaries to further seek domain-invariant new representation. [21] proposed modality-bridge transfer learning for modality classification, which aims at transferring the knowledge acquired from one medical domain to another. [28] passes features extracted by conventional methods to a discrete Bayesian network classifier for hierarchical modality classification. This approach can be used in our models to enable transfer learning for the medical domain models.

III. Approach

Our approach first starts with exploring various services offered by cloud providers and practically implementing deployments of deep learning models on to multiple clouds. We will register common dependencies needed for hosting a model across various cloud providers. We will develop a price catalogue for a optimized deployment on to various cloud providers and choose a algorithm to predict the best option for the deployment. In order for the container to be intelligent in providing inferences for a different domain or task , it will automatically initiate the transfer learning with the most appropriate models available online.

We will use terraform tool to initiate a deployment of the deep learning model across various cloud providers. We will adopt docker containers to containerize the deep learning model developed in popular frameworks. We will develop a python based dynamic catalogue that will pull the most updated information about the pricing options for on demand hosting of the containers. We will explore various hosted ML services and compare it with our custom terraform solution to track the dependencies.

To enable the transfer learning for the containers, more research is needed. We will explore Tensorflow extended to learn about the dynamic refreshing of the model to the streaming data once deployed in production. We will consider the option of going through the source code to come up with a solution.

Our approach starts with identifying the dependencies that are encountered while performing deployment with widely used tools; we then propose a solution that create dynamic environments based on the framework adopted for developing the deep learning model.

A. Dependencies identification

In order to identify the challenges in deploying a deep learning model from local environment , we adopted popular tools like docker and kubernetes ; they are widely used to deploy micro services in industry now. We deployed deep learning models developed using popular frameworks like keras, pytorch, caffee as docker containers on to the kubernetes platform. In order to containerize, we have developed a dockerfile with python 3 base image ; built the docker image and deployed it using kubectl cli commands on to the kubernetes cluster. We realized the dependencies that are needed to build the docker image varies based on the framework libraries and server environement provisioing. Most of the deep learning models are currently wrapped using flask framework. The challenging part is to dynamically recognize the framework libraries that need to be included in the base container. That leads to dynamic creation of dockerfile with those dependencies included. We are proposing a solution that can recognize the dependencies and generate a dynamic dockerfile based on the dependencies. By dynamically generating the dockerfile based on the framework libraries recognized during runtime will help in creation of dynamic docker containers.

B. Dependencies mapping

Our approach should not be contraint by the ever changing version that are released. We want to include a dependency mapping functionality that can map the current environment library versions are all compatible for the deep learning model. This mapping is updated dynamically based on the correct combination of the dependecnies needed. Our approach will detect the best combination of library versions that can lead to a successful creation of docker image. The implementation takes in to consideration the many combinations of versions that are compatible. We want to implement a machine learning model that can automate the process of choosing the best working version combination to populate the docker file.

C. Methodology

We created a proof of concept to gather information about the common dependencies in containerizing a keras based deep learning model. We created a requirements file that will include all the dependencies needed for the model.We made use of flask framework to deploy the model as a restful api. We used python3.6 base docker image and installed keras framework through pip, we made use of environmental variable to set the version of keras to be installed in the container. Docker engine uses the docker file to generate a docker image.

Listing 1. Dockerfile

```
FROM python:3.6
WORKDIR /app
COPY predict.py /app
COPY model_file.h5 /app
COPY requirements.txt /app
RUN pip install -r ./requirements.txt
ARG KERAS_VERSION=2.2.5
CMD ["python3", "predict.py"]
```

Listing 2. predict.py

```
from keras.models import load_model
import numpy as np
import cv2
import sys
from operator import itemgetter
import tensorflow as tf
from flask import Flask, render_template
import flask

from PIL import Image
import io

app = Flask(__name__)
model = None

#Predict Function
@app.route("/predict", methods=["POST"])
def xpredict( image_path):

   # predict function goes here

if __name__ == "__main__":

        app.run(host='0.0.0.0')
```

We have observed from listing 2 that the flask run should be on 0.0.0.0 than local or 127.0.0.1 , this change will allow the flask application in the container which can be served on any public or private cloud. Some of the important challenges are build failures due to dependencies conflicts and we observed that to create dynamic docker container creation , we should generate dynamic docker file that can be used to create a docker image. Through dependency mapping values , we have desgined a system that can generate the dockerfile based on the model framework dependencies.

We are also consider a option to use ONNX framework to port any framework to a framework that is more compatible in creating docker images on the fly. This will resolve complexities in the dependencies mapping approach. ONNX format can be used to port model frameworks to frameworks which are container friendly. We are currently using a python script to build and push the container to the kubernetes cluster. We have observed network latencies in pushing the model to the cluster due to sheer size of the docker image.

We have found a similar implementation of the dynamic container creation from a project called deepo. This open source project contains Dockerfile generator using python. They have modules defined to build the custom docker images. We want to implement the similar approach with automatic customisation of the docker images based on the source code.

Psuedo code for creation of container

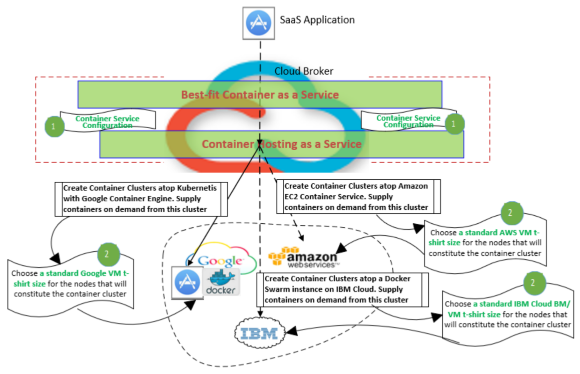
SaaS Application
Cloud Broker
Best-fit Container as a Service
Container Service Configuration
1
Container Service Configuration
1
Container Hosting as a Service
Create Container Clusters atop Kubernetis with Google Container Engine. Supply containers on demand from this cluster
Create Container Clusters atop Amazon EC2 Container Service. Supply containers on demand from this cluster
2
Choose a standard AWS VM t-shirt size for the nodes that will constitute the container cluster
2
Choose a standard Google VM t-shirt size for the nodes that will constitute the container cluster
Google
docker
amazon webservices
Create Container Clusters atop a Docker Swarm instance on IBM Cloud. Supply containers on demand from this cluster
2
Choose a standard IBM Cloud BM/ VM t-shirt size for the nodes that will constitute the container cluster
IBM

Fig. 1. Dynamic container deployment to Public cloud

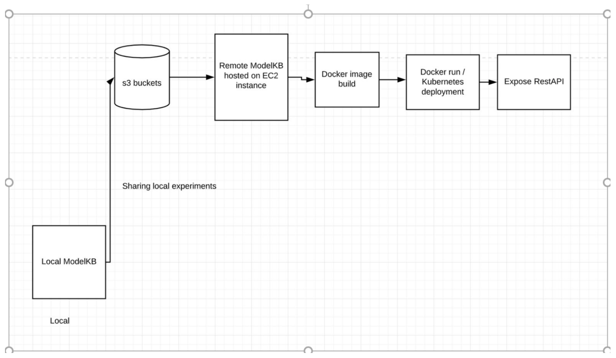
s3 buckets
Remote ModelKB hosted on EC2 instance
Docker image build
Docker run / Kubernetes deployment
Expose RestAPI
Sharing local experiments
Local ModelKB
Local

Fig. 2. Sharing containerized of deep learning model

We containerized a Deep learning model that can classify fashion MNIST dataset After model training is done on local machine , the experiment is shared to remote ModelKB cloud repository.We used AWS S3 object storage as the backend for the Cloud repository which stores the model hdf5 file and inference function.We added boilerplate code to wrap the inference function in a flask framework so that it can expose the model as a Rest API for inferenceOnce the object storage receives the model file, create api service integrated to ModelKB remote platform will trigger a build process. We used Dockerfile with conda base image to install the dependencies like framework and python packages.Docker image is built from the Dockerfile and it is deployed on the AWS EKS cluster as a service. The service endpoint is exposed through loadbalancer, we integrated this REST API endpoint to our backend .Our ModelKB remote platform test view can be used to upload the image for inference against the REST API, DL model Docker image size could be very large Dockerfile should be optimized to choose the best docker base image. Downloading and uploading large docker images can be time consuming and not very secure. Generating dynamic Dockerfile depending on the framework used can be challenging.Dependencies version mismatch can break the docker image build step. Dynamic tracking of dependency versions that can lead to a successful build.

IV. Conclusion

We projected the implementation of dynamic container for deploying the deep learning models as Restful API. With container orchestration tools like Kubernetes , the containerized deep learning model can be deployed as micro services. Microservice architecture for multiple deep learning models can enable transfer learning ability between models in the cloud ot the edge. We proposed a methodology that can dynamically create containers for portable deep learning models. We want to also project the importance of container environment for the edge which can be lite weight and optimized for inferencing.

References

[1] R. Jaluka, D. Miliksetian, and M. Gupta, Enterprise it as a service: Transforming the delivery model of it services, in IEEE International Conference on Cloud Computing for Emerging Markets, 2016.

[2] . von Laszewski, J. Diaz, F. Wang, and G. C. Fox, Comparison of multiple cloud frameworks, in 2012 IEEE Fifth International Conference on Cloud Computing. IEEE, jun 2012.

[3] S. Shirinbab, L. Lundberg, and E. Casalicchio, Performance eval- uation of container and virtual machine running cassandra work- load, in International Conference Cloud Computing Technologies and Applications, 2017.

[4] S. Sebastio, R. Ghosh, and T. Mukherjee, An availability analy- sis approach for deployment configurations of containers, IEEE Transactions on Services Computing, pp. 11, 2018.

[5] S.Nadgowda,S.Suneja,andA.Kanso,Comparingscalingmeth- ods for linux containers, in IEEE International Conference on Cloud Engineering (IC2E). IEEE, apr 2017.

[6] Y. Mao, J. Oak, and A. Pompili, Draps: Dynamic and resource- aware placement scheme for docker containers in a heterogeneous cluster, in IEEE Performance Computing and Communications Con- ference, 2017

[7] H.Liang,Q.Hao,andM.Li,Semantics-based anomaly detection of processes in linux containers, in International Conference on Identification, Information and Knowledge in the Internet of Things, 2016.

[8] G. Tricomi, A. Panarello, G. Merlino, F. Longo, D. Bruneo, and A. Puliafito, Orchestrated multi-cloud application deployment in openstack with tosca, in IEEE International Conference on Smart Computing, 2017

[9] T.Lynn,P.Rosati,A.Lejeune,andV.Emeakaroha,Apreliminary review of enterprise serverless cloud computing platforms, in IEEE International Conference on Cloud Computing Technology and Science, 2017.

[10] Evan R. Sparks, Shivaram Venkataraman, Tomer Kaftan, Michael J. Franklin, and Benjamin Recht. 2016. KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics. CoRR abs/1610.09451 (2016).

[11] R. Jaluka, D. Miliksetian, and M. Gupta, Enterprise it as a service: Transforming the delivery model of it services, in IEEE International Conference on Cloud Computing for Emerging Markets, 2016.

[12] Domain adaptation models , M. Jiang, W. Huang, Z. Huang, and G. G. Yen, Integration of global and local metrics for domain adaptation learning via dimensionality reduction, IEEE Trans. Cybern., vol. 47, no. 1, pp. 3851, Jan. 2017.

[13] Sreekrishnan Venkateswaran and Santonu Sarkar, "Fitness aware containerization service leveraging machine learning" , IEEE, 2019.

[14] S.Nadgowda,S.Suneja,andA.Kanso, Comparing scaling methods for linux containers, in IEEE International Conference on Cloud Engineering (IC2E). IEEE, apr 2017.

[15] W. Hanafy, A. Mohamed, and S. Salem, Novel selection policies for container-based cloud deployment models, in International Computer Engineering Conference, 2017.

[16] H.Liang,Q.Hao,andM.Li,Semantics-based anomaly detection of processes in linux containers, in International Conference on Identification, Information and Knowledge in the Internet of Things, 2016.

[17] Evan R. Sparks, Shivaram Venkataraman, Tomer Kaftan, Michael J. Franklin, and Benjamin Recht. 2016.

[18] Sara Landset, Taghi M. Khoshgoftaar, Aaron N. Richter, and Tawfiq Hasanin. 2015. A survey of open source tools for machine learning with big data in the Hadoop ecosystem. Journal of Big Data 2, 1 (2015), 24.

[19] N. D. Lane, P. Georgiev, and L. Qendro, Deepear: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments Using Deep Learning, Proc. 2015 ACM Intl. Joint Conf. Pervasive and Ubiquitous Computing, 2015, pp. 28394.

[20] Alsheikh et al. Mobile Big Data Analytics Using Deep Learning and Apache Spark, IEEE Network, vol. 30, no. 3, May/June 2016, pp. 2229.

[21] K. Messer, W. Christmas, J. Kittler, "Automatic sports classification", Proc. IEEE Int. Conf. Pattern Recognition, pp. 1005-1008, 2002.

[22] Z. Ding, M. Shao, and Y. Fu, Robust multi-view representation: A unified perspective from multi-view learning to domain adaption, in Proc. Int. Joint Conf. Artif. Intell., 2018, pp. 54345440.

[23] S. Shekhar, V. M. Patel, H. V. Nguyen, and R. Chellappa, Generalized domain-adaptive dictionaries, in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2013, pp. 361368

[24] H. G. Kim, Y. Choi, and Y. M. Ro, Modality-bridge transfer learning for medical image classification, in International Congress on Image and Signal Processing, BioMedical Engineering and Informatics, 2017, pp. 15.

[25] J. Arias, J. Martnez-Gmez, J. A. Gmez, A. G. S. de Herrera, and H. Mller, Medical image modality classification using discrete Bayesian networks, Computer Vision and Image Understanding, vol. 151, pp. 61 71, 2016.

] L. Li et al., Eyes in the Dark: Distributed Scene Understanding for Disaster Management, IEEE Trans. Parallel Distrib. Systems, 2017. DOI: 10.1109/TPDS.2017.2740294.